§ Ignition Gazebo (Gazebo Sim) Installation

安裝教學網址：https://gazebosim.org/docs/all/getstarted

Platform	Gazebo Versions
Ubuntu 22.04 Jammy	Gazebo Harmonic (recommended), Gazebo Garden and Gazebo Fortress
Ubuntu 20.04 Focal	Gazebo Garden (recommended), Gazebo Fortress and Gazebo Citadel
Ubuntu 18.04 Bionic	Gazebo Citadel
Mac Monterey	Gazebo Harmonic (recommended), Gazebo Garden, Gazebo Fortress and Gazebo Citadel
Mac BigSur	Gazebo Garden (recommended), Gazebo Fortress and Gazebo Citadel
Windows	Support via Conda-Forge is not fully functional, as there are known runtime issues see this issue for details.

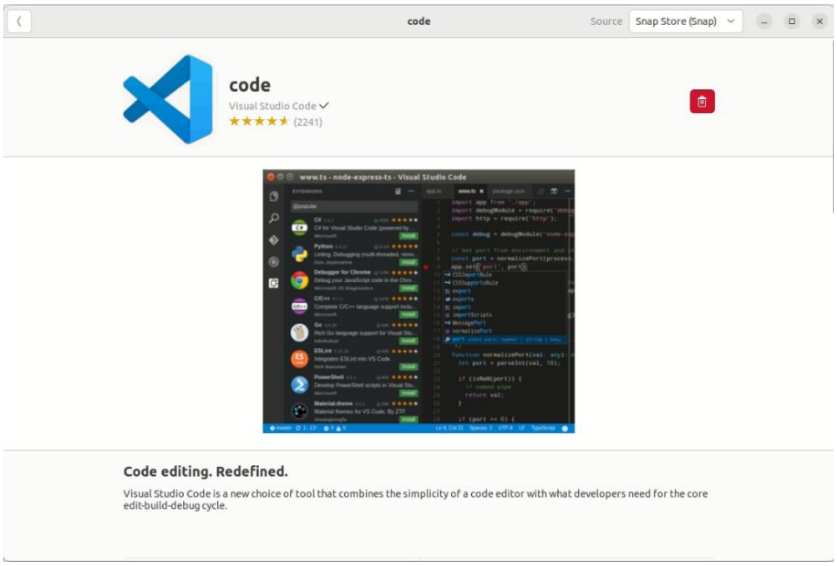
請按照作業系統版本安裝，否則會裝得了但打不開

此篇為使用 Ubuntu 20.04, Gazebo Garden

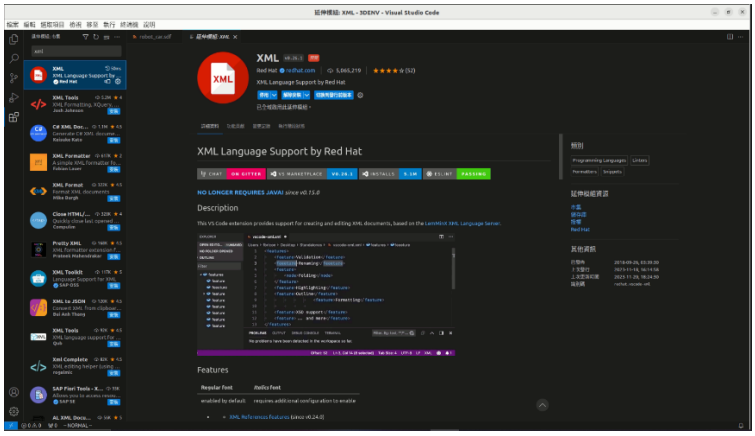
安裝完成請使用`$gz sim` 或是官網指令測試是否能打開

§ Visual Studio Code

請安裝 VSCode 來編寫 SDF 檔，可在 Ubuntu Software 商店中安裝。



並在 VSCode 的延伸模組商店中安裝 XML：



§ Creating worlds

Defining a world:

每個 SDF 檔的開始會有些 tags：

```
<?xml version="1.0" ?>
<sdf version="1.8">
    <world name="world_demo">
    ...
    ...
    </world>
</sdf>
```

前兩者分別定義了 xml 與 sdf 的版本，</sdf>則表示了此 sdf 檔的結束。世界的所有都會在 world tag(<world>與</world>)中間進行，此範例的 world 取名為”world_demo”。

Adding physics to the world:

```
<physics name="1ms" type="ignored">
    <max_step_size>0.001</max_step_size>
    <real_time_factor>1.0</real_time_factor>
</physics>
```

<physics>與</physics>框住物理定義，此 physics 名稱命名為”1ms”，屬性(type)為使之物理引擎(Dynamic engine)，可選擇 Dart, Ode, Bullet, Simbody……，設為”ignored”是因為物理引擎還不是通過這個 tag 來選擇的。

<max_step_size>是模擬的系統可以與世界(state of the world)互動的最大時間，數值越小越計算越準確，但同時也需要更多的運算資源。[s]

<real_time_factor>是模擬時間與實際時間的比例。

Adding Plugins to the world:

Plugin 是一個動態加載的程式碼塊(Dynamic load chunk of code)，例：

```
<plugin
    filename="gz-sim-physics-system"
    name="gz::sim::systems::Physics">
</plugin>
```

Physics plugin 對於動態模擬十分重要

```
<plugin
    filename="gz-sim-user-commands-system"
    name="gz::sim::systems::UserCommands">
</plugin>
```

User commands plugin 負責創建模型、移動模型、刪除模型與其他使用者指令。

```
<plugin
    filename="gz-sim-scene-broadcaster-system"
    name="gz::sim::systems::SceneBroadcaster">
</plugin>
```

Scene broad caster plugin 顯示世界的場景。

Gui tag:

範例：

```
<gui fullscreen="0">
    ...
    ...
</gui>
```

以下的 Gui 配置均在上方範例中的"..."進行。

gz-gui 有多個可供選擇的插件，以下將添加一些必要插件使世界能運行基本功能。

```
<!-- 3D scene -->
<plugin filename="MinimalScene" name="3D View">
  <gz-gui>
    <title>3D View</title>
    <property type="bool" key="showTitleBar">false</property>
    <property type="string" key="state">docked</property>
  </gz-gui>

  <engine>ogre2</engine>
  <scene>scene</scene>
  <ambient_light>0.4 0.4 0.4</ambient_light>
  <background_color>0.8 0.8 0.8</background_color>
  <camera_pose>-6 0 6 0 0.5 0</camera_pose>
  <camera_clip>
    <near>0.25</near>
    <far>25000</far>
  </camera_clip>
</plugin>
<plugin filename="GzSceneManager" name="Scene Manager">
  <gz-gui>
    <property key="resizable" type="bool">false</property>
    <property key="width" type="double">5</property>
    <property key="height" type="double">5</property>
    <property key="state" type="string">floating</property>
    <property key="showTitleBar" type="bool">false</property>
  </gz-gui>
</plugin>
```

MinimalScene 和 GzSceneManager 負責顯示 3D 場景

showTitleBar 會在 plugin 上方顯示藍色標題欄，其標題名稱則為該 plugin 名稱。

state 是插件狀態，可用 docked 固定在定點，或是用 floating 浮動

渲染引擎可選 ogre 或 ogre2

ambient_light 和 background_color 負責場景的環境光與背景顏色，camera_pose 負責相機的 x, y, z 位置

```
<!-- World control -->
<plugin filename="WorldControl" name="World control">
    <gz-gui>
        <title>World control</title>
        <property type="bool" key="showTitleBar">false</property>
        <property type="bool" key="resizable">false</property>
        <property type="double" key="height">72</property>
        <property type="double" key="width">121</property>
        <property type="double" key="z">1</property>

        <property type="string" key="state">floating</property>
        <anchors target="3D View">
        <line own="left" target="left"/>
        <line own="bottom" target="bottom"/>
        </anchors>
    </gz-gui>

    <play_pause>true</play_pause>
    <step>true</step>
    <start_paused>true</start_paused>
    <service>/world/world_demo/control</service>
    <stats_topic>/world/world_demo/stats</stats_topic>
</plugin>
```

World control 插件負責控制世界

play_pause 負責顯示左下角的暫停撥放按鈕，如下圖所示：

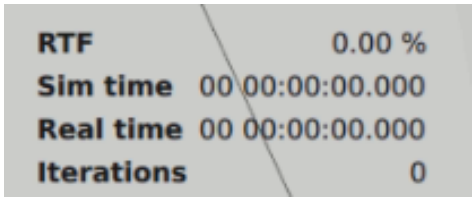
stats_topic 負責世界訊息統計(模擬時間與實際時間)的標題

start_paused 若為 true 則 Gazebo 啟動時預設為暫停模擬狀態。

```
<!-- World statistics -->
<plugin filename="WorldStats" name="World stats">
    <gz-gui>
        <title>World stats</title>
        <property type="bool" key="showTitleBar">false</property>
        <property type="bool" key="resizable">false</property>
        <property type="double" key="height">110</property>
        <property type="double" key="width">290</property>
        <property type="double" key="z">1</property>

        <property type="string" key="state">floating</property>
        <anchors target="3D View">
        <line own="right" target="right"/>
        <line own="bottom" target="bottom"/>
        </anchors>
    </gz-gui>

    <sim_time>true</sim_time>
    <real_time>true</real_time>
    <real_time_factor>true</real_time_factor>
    <iterations>true</iterations>
    <topic>/world/world_demo/stats</topic>

</plugin>
```

WorldStats 插件負責顯示世界訊息，包含 sim_time(模擬時間)、real_time(實際時間)、real_time_factor(模擬、實際時間比例)、iterations(跌代)。

這些標籤可以讓我們選擇在介面右下角要顯示的值，如下圖所示：



也可透過開啟另一個終端機來監看 stats：

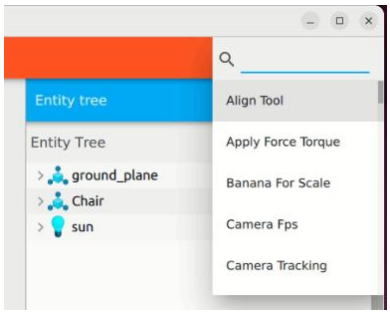
1. 執行世界：$ gz sim world_demo.sdf
2. 按左下角的撥放按鈕開始模擬
3. 開啟另一個終端機(需相同目錄下)：$ gz topic -e -t /world/world_demo/stats

註：/world/car_world/stats 為 WorldStats plugin 中所設的 topic 名稱

```
<!-- Entity tree -->
<plugin filename="EntityTree" name="Entity tree">
</plugin>
```

實體樹插件可以有一個欄位顯示目前世界中所有的物件。

還有許多有用的 gz-gui 插件，也能從軟體插件選單中下拉添加，如下圖：



Light:

```
<light type="directional" name="sun">
    <cast_shadows>true</cast_shadows>
    <pose>0 0 10 0 0 0</pose>
    <diffuse>0.8 0.8 0.8 1</diffuse>
    <specular>0.2 0.2 0.2 1</specular>
    <attenuation>
        <range>1000</range>
        <constant>0.9</constant>
        <linear>0.01</linear>
        <quadratic>0.001</quadratic>
    </attenuation>
    <direction>-0.5 0.1 -0.9</direction>
</light>
```

Light tag 指定了世界的光源，光源的 type 可以是 point(點光源)、directional(方向光源)、spot(聚光燈)。

cast_shadow 為 true 時，會產生陰影。

diffuse 和 specular 是漫射和鏡面光的顏色。

pose 是光元素的相對位置(x, y, z, roll, pitch, yaw)通常被忽略，因為相對於世界

attenuation 指定光的衰減屬性，包含

1. range 光的範圍
2. constant 衰減因子，1 代表不衰減，0 代表完全衰減
3. linear 線性衰減因子，1 代表均勻衰減
4. quadratic 二次衰減因子，對衰減添加曲率
5. direction 光的方向，僅適用於 directional 與 spot

Adding model:

可到 Gazebo fuel 網站中載入一些模型：https://app.gazebosim.org/fuel

載入方式 1：

挑選一個模型並且點選< > copy SDF snippet to clipboard，如下圖：

並將他貼在 world 標籤內。

載入方式 2：

先將 GZ_SIM_RESOURCE_PATH 設在你存放.sdf 檔的目錄，假設.sdf 放在家目錄底下的 3DENV 目錄，則下指令如下：

```
$ export GZ_SIM_RESOURCE_PATH="$HOME/3DENV"
```

接著將 fuel 上的 model 下載下來解壓縮後放在與.sdf 相同目錄

再來將此段加進 world tag 內(model 名以 Coke 為例)：

```
<include>
    <uri>
    model://Coke
    </uri>
</include>
```

也可在 SDF 內改變模型位置(加上<pose>)，如下所示：

```
<include>
    <name>Coke1</name>
    <pose>0 0.1 0 0 0 0</pose>
    <uri>https://fuel.gazebosim.org/1.0/OpenRobotics/models/Coke</uri>
</include>
```

§ Build a simple robot car

Building a simple world:

參照上節先建立一個基本的世界，最後建立一個 model 來描述地板，如下所示：

```
<model name="ground_plane">
    <static>true</static>
    <link name="link">
        <collision name="collision">
        <geometry>
            <plane>
            <normal>0 0 1</normal>
            </plane>
        </geometry>
        </collision>
        <visual name="visual">
        <geometry>
            <plane>
            <normal>0 0 1</normal>
            <size>100 100</size>
            </plane>
        </geometry>
        <material>
            <ambient>0.8 0.8 0.8 1</ambient>
            <diffuse>0.8 0.8 0.8 1</diffuse>
            <specular>0.8 0.8 0.8 1</specular>
        </material>
        </visual>
    </link>
</model>
```

Link tag 用來表達物件，visual tag 則是表達在軟體介面中外觀。需特別注意的是 collision tag 用來表達可碰撞參數，如未表達 collision 則物體將無法與物體間有接觸。

Building the model:

定義模型如下：

```
<model name='vehicle_blue' canonical_link='chassis'>
    <pose relative_to='world'>0 0 0 0 0 0</pose>
```

名稱為'vehicle_blue'，每個模型可以指定一個 link(桿件)作為 canonical_link(類似主要桿件)，若未指派則會默認第一個 link 為 canonical_link。

pose tag 可選擇相對於其他物件的位置，參數為：X Y Z R(Roll) P(Pitch) Y(Yaw)

Links forming the robot:

每個 model 都是由 joints 連結在一起的 link(s)所組成的

1. Chassis(車架)

```
<link name='chassis'>
    <pose relative_to='__model__'>0.5 0 0.4 0 0 0</pose>
```

先定義 chassis 與其位置，接著定義其 inertial properties，如下：

```
<inertial> <!--inertial properties of the link mass, inertia matix-->
    <mass>1.14395</mass>
    <inertia>
        <ixx>0.095329</ixx>
        <ixy>0</ixy>
        <ixz>0</ixz>
        <iyy>0.381317</iyy>
        <iyz>0</iyz>
        <izz>0.476646</izz>
    </inertia>
</inertial>
```

這邊提供 inertia 計算機：click_me

接著定義 visual：

```
<visual name='visual'>
    <geometry>
        <box>
            <size>2.0 1.0 0.5</size>
        </box>
    </geometry>
    <!--let's add color to our link-->
    <material>
        <ambient>0.0 0.0 1.0 1</ambient>
        <diffuse>0.0 0.0 1.0 1</diffuse>
        <specular>0.0 0.0 1.0 1</specular>
    </material>
</visual>
```

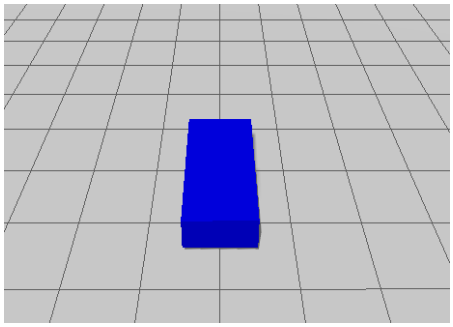
geometry tag 中選擇將使用的形狀，與其 size(x, y, z)，在 material tag 中定義顏色(red, green, blue, alpha)，各顏色範圍值介於 0~1。

最後是定義 collision：

```
      <collision name='collision'>
          <geometry>
              <box>
                  <size>2.0 1.0 0.5</size>
              </box>
          </geometry>
      </collision>
  </link>
</model>
```

註：collision 大小不一定要與外觀大小相同，通常會依情況最小化 collision 以減少計算量。

最後的 model 在介面中應如下圖所示：



2. Left wheel(左輪)

添加在車子的 model tag 內：

```
<link name='left_wheel'>
    <pose relative_to="chassis">-0.5 0.6 0 -1.5707 0 0</pose>
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.043333</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.043333</iyy>
            <iyz>0</iyz>
            <izz>0.08</izz>
        </inertia>
    </inertial>
```

與 Chassis 建立相似，但在位置的 pitch 要調整為-1.5707(rad)，也就是-90 度，來使車輪是立起來的狀態。

同樣的需要建立 visual 與 collision：

```
    <visual name='visual'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
        <material>
            <ambient>1.0 0.0 0.0 1</ambient>
            <diffuse>1.0 0.0 0.0 1</diffuse>
            <specular>1.0 0.0 0.0 1</specular>
        </material>
    </visual>
    <collision name='collision'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
    </collision>
</link>
```

所使用的形狀為 cylinder，故需要 radius 與 length 兩個參數。

3. Right wheel(右輪)

   與左輪相似：

```
<!--The same as left wheel but with different position-->
<link name='right_wheel'>
    <pose relative_to="chassis">-0.5 -0.6 0 -1.5707 0 0</pose> <!--angles are in radian-->
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.043333</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.043333</iyy>
            <iyz>0</iyz>
            <izz>0.08</izz>
        </inertia>
    </inertial>
    <visual name='visual'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
        <material>
            <ambient>1.0 0.0 0.0 1</ambient>
            <diffuse>1.0 0.0 0.0 1</diffuse>
            <specular>1.0 0.0 0.0 1</specular>
        </material>
    </visual>
    <collision name='collision'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
    </collision>
</link>
```

4. Defining an arbitrary frame

   新增一個前輔助輪的框架：

```
<frame name="caster_frame" attached_to='chassis'>
    <pose>0.8 0 -0.2 0 0 0</pose>
</frame>
```

   註：這邊沒有用到 relative_to 是因為已經用了 attached_to，兩者有相同的功能。

5. Caster wheel(腳輪/輔助輪)

```
<!--caster wheel-->
<link name='caster'>
    <pose relative_to='caster_frame'/>
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.016</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.016</iyy>
            <iyz>0</iyz>
            <izz>0.016</izz>
        </inertia>
    </inertial>
    <visual name='visual'>
        <geometry>
            <sphere>
                <radius>0.2</radius>
            </sphere>
        </geometry>
        <material>
            <ambient>0.0 1 0.0 1</ambient>
            <diffuse>0.0 1 0.0 1</diffuse>
            <specular>0.0 1 0.0 1</specular>
        </material>
    </visual>
    <collision name='collision'>
        <geometry>
            <sphere>
                <radius>0.2</radius>
            </sphere>
        </geometry>
    </collision>
</link>
```

與先前定義左右輪情形類似，不同的是相對位置這次是相對於前點所建立的輔助輪框架，且形狀用的是 sphere(所需參數為 radius)。

Joints(connecting links together):

Joint tag 負責將桿件連結在一起，並定義桿件之間相對運動的方式

1. Left wheel joint

```
<joint name='left_wheel_joint' type='revolute'>
    <pose relative_to='left_wheel'/>
```

定義 joint 的名稱、運動模式(type)，與其連結位置。所使用的 type 為 revolute，提供了 1 DOF 的純轉動。

```
<parent>chassis</parent>
<child>left_wheel</child>
```

每個 joint 會連結兩個桿件，所以此處得定義子母桿件。

```
    <axis>
        <xyz expressed_in='__model__'>0 1 0</xyz> <!--can be defined as any frame or even arbitrary frames-->
        <limit>
            <lower>-1.79769e+308</lower>    <!--negative infinity-->
            <upper>1.79769e+308</upper>     <!--positive infinity-->
        </limit>
    </axis>
</joint>
```

axis tag 定義了旋轉軸，輪胎相對於 model 是在 y 軸旋轉，故給予 0,1,0 的參數值，並且在 limit tag 中定義極限旋轉角度，由於輪胎回一直轉，故定義為±無限大(in rad)。

2. Right wheel joint

與 Left wheel joint 相同，但須將位置、子母桿件做更換：

```
<joint name='right_wheel_joint' type='revolute'>
    <pose relative_to='right_wheel'/>
    <parent>chassis</parent>
    <child>right_wheel</child>
    <axis>
        <xyz expressed_in='__model__'>0 1 0</xyz>
        <limit>
            <lower>-1.79769e+308</lower>    <!--negative infinity-->
            <upper>1.79769e+308</upper>     <!--positive infinity-->
        </limit>
    </axis>
</joint>
```

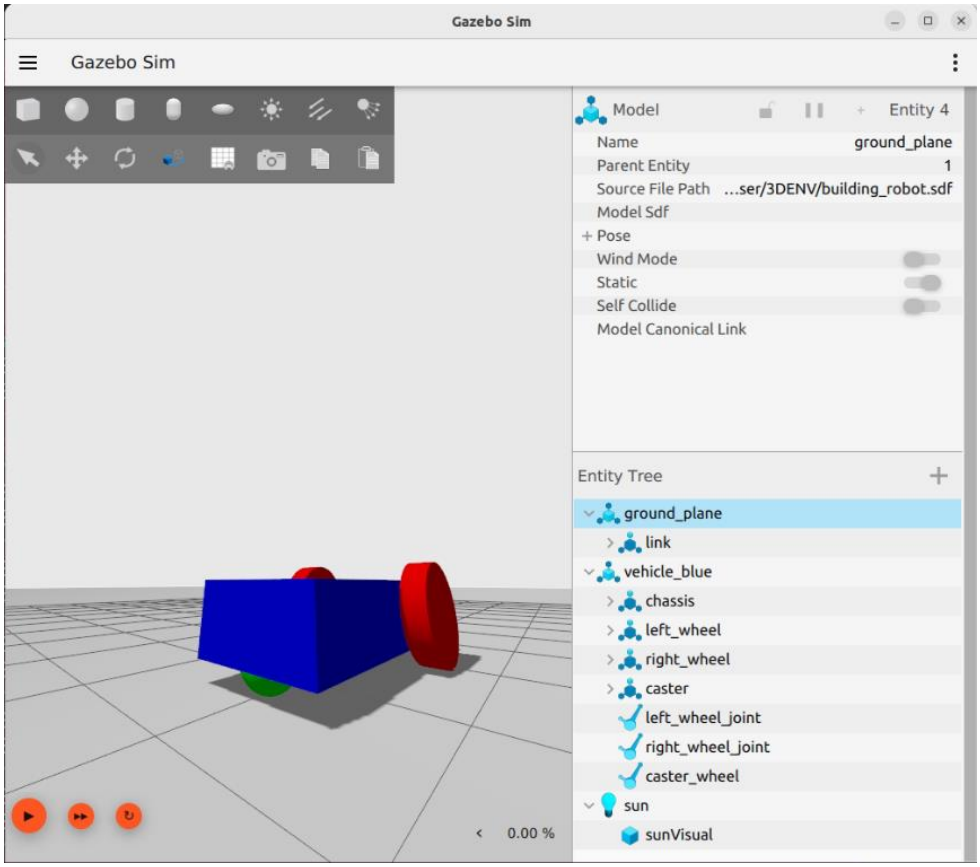
3. Caster wheel joint

```
<joint name='caster_wheel' type='ball'>
    <parent>chassis</parent>
    <child>caster</child>
</joint>
```

type 指定為 ball，即給予 3 DOF 的轉動。

Conclusion:

結果應會如下圖所示：



在 Entity Tree 中可以看到所以建立的 links 與 joints。

§ Moving the robot car

Diff_drive plugin:

diff_drive 插件可以用來移動車子，將以下代碼放入要移車子的 model tag 內：

```
<plugin
    filename="gz-sim-diff-drive-system"
    name="gz::sim::systems::DiffDrive">
    <left_joint>left_wheel_joint</left_joint>
    <right_joint>right_wheel_joint</right_joint>
    <wheel_separation>1.2</wheel_separation>
    <wheel_radius>0.4</wheel_radius>
    <odom_publish_frequency>1</odom_publish_frequency>
    <topic>cmd_vel</topic>
</plugin>
```

left_joint 與 right_joint 分別放入車子定義之左、右 joint。

在上節定義車子時，輪子的 y 軸分別是 0.6 與-0.6，故 wheel_separation 為 1.2。

wheel_radius 取輪子半徑(已於 model tag 中定義過了)。

odom_publish_frequency 是設置里程計發布到/model/vehicle_blue/odometry 的頻率。

cmd_vel 是 diff_drive 插件的輸入 topic。

Topics and Messages:

插入上述插件後，即可用終端機移動模型車。首先從終端機打開模擬(先別按撥放鍵，接著新增一個終端機，輸入指令如下：

```
$ gz topic -t "/cmd_vel" -m gz.msgs.Twist -p "linear: {x: 0.5}, angular: {z: 0.05}"
```

-t 指定要發布之 topic 名稱，-m 指定消息類型。以此範例為例，在模擬介面中按下撥放鍵後，車子將向 x 軸以 0.5 m/s 的速度移動，並同時以 z 軸為旋轉軸，角速度為 0.05 rad/s 轉動。可使用以下指令來了解每個 topic 的作用：

`$ gz topic -h`。

Moving the robot using the keyboard:

除了用指令輸入以外，也能透過鍵盤輸入。若要透過鍵盤輸入則需要兩個插件：KeyPublisher 與 TriggerPublisher。

1. KeyPublisher：

   此插件可以幫助你找到在模擬中按下鍵盤所對應到的代碼是多少。

   在 gui tag 中加入`<plugin filename="KeyPublisher" name="Key Publisher"/>`，

   接著用終端機打開模擬後再新增一個終端機，輸入以下指令：

   `$ gz topic -e -t /keyboard/keypress`

   接著在軟體介面中按下撥放鍵，並且隨意按下鍵盤按鍵，終端機就會跳出相對應的數字代碼，以下分別為上、下、左、右方向鍵的例子：

   ```
   data: 16777235

   data: 16777237

   data: 16777234

   data: 16777236
   ```

2. TriggeredPublisher

此插件可以指定任意 topic 作為令一 topic 的輸入，在此我們將指定鍵盤輸入的 topic(/keyboard/keypress)作為移動車子的 topic(cmd_vel)的輸入。以下將以上方向鍵作為前進的訊號，在 world tag 中加入以下代碼：

```
<!-- Moving Forward-->
<plugin filename="gz-sim-triggered-publisher-system"
        name="gz::sim::systems::TriggeredPublisher">
    <input type="gz.msgs.Int32" topic="/keyboard/keypress">
        <match field="data">16777235</match>
    </input>
    <output type="gz.msgs.Twist" topic="/cmd_vel">
        linear: {x: 0.5}, angular: {z: 0.0}
    </output>
</plugin>
```

這裡指定了 input 與 output，其中 16777235 代表的就是上方向鍵。

以上為例也可用下、左、右鍵分別建立向後、左旋轉、右旋轉的 plugin：

```
<!-- Moving Backward-->
<plugin filename="gz-sim-triggered-publisher-system"
        name="gz::sim::systems::TriggeredPublisher">
    <input type="gz.msgs.Int32" topic="/keyboard/keypress">
        <match field="data">16777237</match>
    </input>
    <output type="gz.msgs.Twist" topic="/cmd_vel">
        linear: {x: -0.5}, angular: {z: 0.0}
    </output>
</plugin>
```

```
<!-- Rotating Left-->
<plugin filename="gz-sim-triggered-publisher-system"
        name="gz::sim::systems::TriggeredPublisher">
    <input type="gz.msgs.Int32" topic="/keyboard/keypress">
        <match field="data">16777234</match>
    </input>
    <output type="gz.msgs.Twist" topic="/cmd_vel">
        linear: {x: 0.0}, angular: {z: 0.5}
    </output>
</plugin>
```

```
<!-- Rotating Right-->
<plugin filename="gz-sim-triggered-publisher-system"
        name="gz::sim::systems::TriggeredPublisher">
    <input type="gz.msgs.Int32" topic="/keyboard/keypress">
        <match field="data">16777236</match>
    </input>
    <output type="gz.msgs.Twist" topic="/cmd_vel">
        linear: {x: 0.0}, angular: {z: -0.5}
    </output>
</plugin>
```

最後在軟體介面中按下撥放鍵就能透過鍵盤控制車子了。

§ Sensors

IMU Sensor:

IMU sensor 的 plugin 可以輸出物件的方向、角速度與直線加速度，建立 IMU plugin 在 world tag 內如下：

```
<plugin filename="gz-sim-imu-system"
        name="gz::sim::systems::Imu">
</plugin>
```

接著將 IMU plugin 以 sensor tag 的形式建立在 chassis 的 link tag 底下：

```
<sensor name="imu_sensor" type="imu">
    <always_on>1</always_on>
    <update_rate>1</update_rate>
    <visualize>true</visualize>
    <topic>imu</topic>
</sensor>
```

always_on 若為 true 則會依據 update_rate 更新數據

visualize 定義是否顯示 sensor GUI 介面在軟體上

topic 定義發布訊息的名稱

若要查看 IMU data 可以依照以下方式進行：

1. 使用終端機打開模擬，並點選左下角撥放鍵開始模擬
2. 開啟另一個終端機輸入指令：`$ gz topic -e -t /imu`

接著終端機就會顯示物件的方向、角速度與直線加速度如下圖：

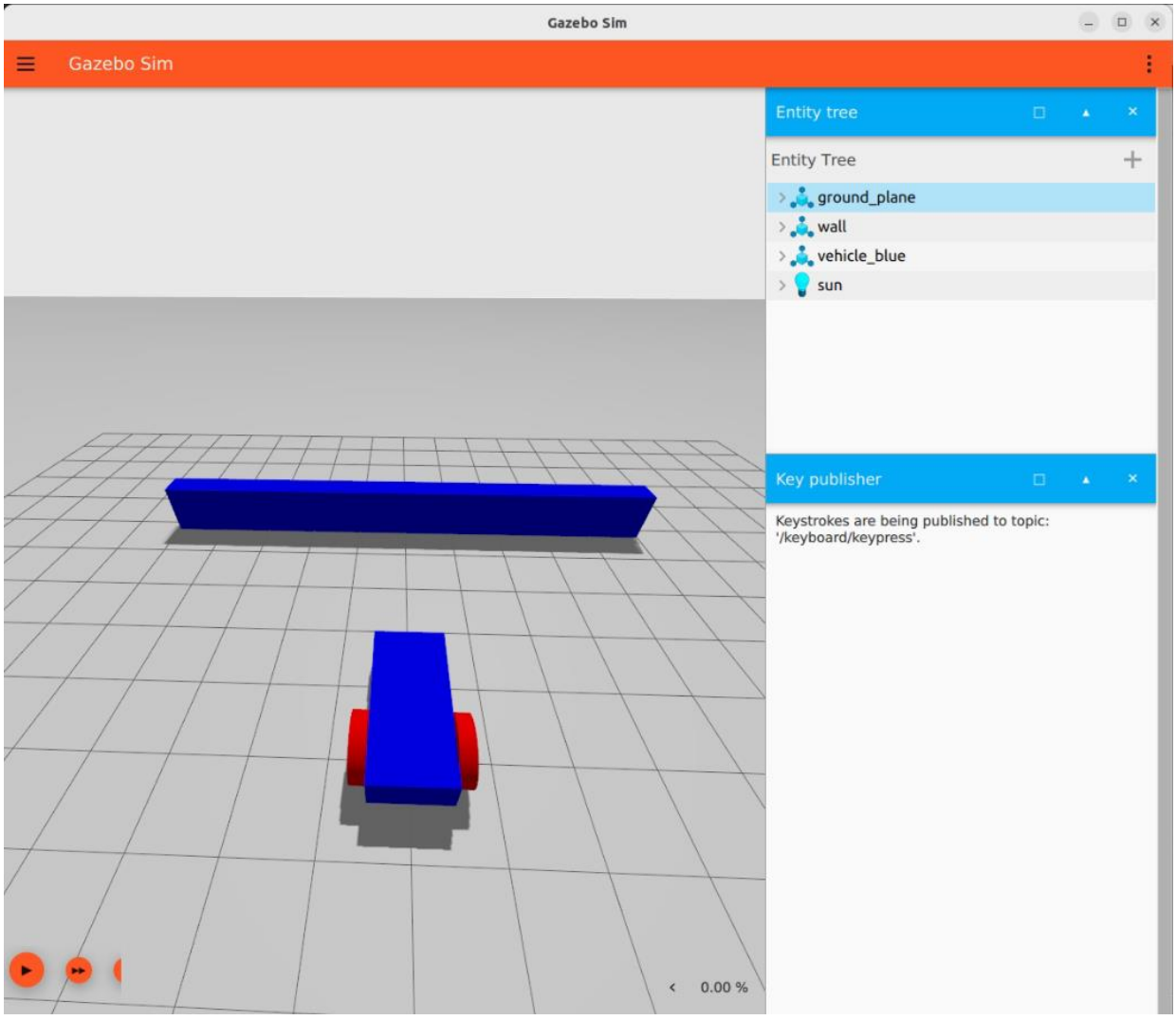
```
header {
  stamp {
    sec: 22
  }
  data {
    key: "frame_id"
    value: "vehicle_blue::chassis::imu_sensor"
  }
  data {
    key: "seq"
    value: "19"
  }
}
entity_name: "vehicle_blue::chassis::imu_sensor"
orientation {
  x: 9.5539749825751779e-12
  y: 3.7041665141327274e-06
  z: 6.9474905014058687e-07
  w: 0.99999999999289824
}
angular_velocity {
  x: 4.5337979445223294e-13
  y: 4.4239080166856279e-13
  z: 1.7907998732677392e-17
}
linear_acceleration {
  x: -7.2601166413169938e-05
  y: -2.8824245862733537e-10
  z: 9.7999999984519413
}
```

Contact Sensor:

此感測器可以讓一物件被另一物間接觸到時發出指示，以下操作將以撞擊牆壁作為例子。先建立一牆壁模型在 world tag 內：

```
<model name='wall'>
    <static>true</static>
    <pose>5 0 0 0 0 0</pose><!--pose relative to the world-->
    <link name='box'>
        <visual name='visual'>
            <geometry>
                <box>
                    <size>0.5 10.0 2.0</size>
                </box>
            </geometry>
            <!--let's add color to our link-->
            <material>
                <ambient>0.0 0.0 1.0 1</ambient>
                <diffuse>0.0 0.0 1.0 1</diffuse>
                <specular>0.0 0.0 1.0 1</specular>
            </material>
        </visual>
        <collision name='collision'>
            <geometry>
                <box>
                    <size>0.5 10.0 2.0</size>
                </box>
            </geometry>
        </collision>
    </link>
</model>
```

如建立成功，開啟模擬後會如下圖所示：



接著將 contact sensor 的 plugin 建立在 world tag 底下：

```
<plugin filename="gz-sim-contact-system"
        name="gz::sim::systems::Contact">
</plugin>
```

接著就能將 contact sensor 的 sensor tag 放在 wall 的 link(box) tag 內：

```
<sensor name='sensor_contact' type='contact'>
    <contact>
        <collision>collision</collision>
    </contact>
</sensor>
```

contact 選擇 collision，則 collision tag 的參數則是綁定 wall model 的 collision 名稱(恰好也是 collision)

此外，需在 wall model tag 內新建一個 Touch plugin，讓牆壁被觸碰時發布訊息：

```
<plugin filename="gz-sim-touchplugin-system"
        name="gz::sim::systems::TouchPlugin">
    <target>vehicle_blue</target>
    <namespace>wall</namespace>
    <time>0.001</time>
    <enabled>true</enabled>
</plugin>
```

target tag 可以選擇被什麼物件觸碰時發布

namespace 決定了發布的 topic 名稱，設置 wall 則對應 topic 即為：/wall/touched

接著測試是否成功：

1. 用終端機打開模擬
2. 開啟另一終端機輸入指令：$ gz topic -e -t /wall/touched
3. 操控車子碰到牆壁，如若成功終端機會顯示：data: true

接著添加 TriggeredPlublisher 插件在 world tag 內，使車子碰到牆壁時停止其速度：

```
<plugin filename="gz-sim-triggered-publisher-system"
        name="gz::sim::systems::TriggeredPublisher">
    <input type="gz.msgs.Boolean" topic="/wall/touched">
        <match>data: true</match>
    </input>
    <output type="gz.msgs.Twist" topic="/cmd_vel">
        linear: {x: 0.0}, angular: {z: 0.0}
    </output>
</plugin>
```

Touchplugin 的輸出 topic(/wall/touched)作為控制車子速度的 topic(/cmd_vel)的輸入，if data: true 則將車子速度、角速度歸零。

Lidar Sensor:

上述插件是有接觸到才讓車子停下來，但有些時候不希望車子碰撞到物體，因此可以使用 Lidar sensor 而非 touch sensor。Lidar 是”light detection and ranging”的縮寫，這種感測器可以檢測機器人周圍的障礙物，以下範例將使用他來量測車子與牆壁的距離，進而避障。

首先須在車子車架處建立一個 frame 來固定 Lidar sensor，在車子的 model tag 中建立：

```
<frame name="lidar_frame" attached_to='chassis'>
    <pose>0.8 0 0.5 0 0 0</pose>
</frame>
```

接著將以下 plugin 建立在 world tag 內：

```
<plugin
  filename="gz-sim-sensors-system"
  name="gz::sim::systems::Sensors">
  <render_engine>ogre2</render_engine>
</plugin>
```

在 chassis link tag 中加入 lidar sensor：

```
<sensor name='gpu_lidar' type='gpu_lidar'>"
    <pose relative_to='lidar_frame'>0 0 0 0 0 0</pose>
    <topic>lidar</topic>
    <update_rate>10</update_rate>
    <ray>
        <scan>
            <horizontal>
                <samples>640</samples>
                <resolution>1</resolution>
                <min_angle>-1.396263</min_angle>
                <max_angle>1.396263</max_angle>
            </horizontal>
            <vertical>
                <samples>1</samples>
                <resolution>0.01</resolution>
                <min_angle>0</min_angle>
                <max_angle>0</max_angle>
            </vertical>
        </scan>
        <range>
            <min>0.08</min>
            <max>10.0</max>
            <resolution>0.01</resolution>
        </range>
    </ray>
    <always_on>1</always_on>
    <visualize>true</visualize>
</sensor>
```

sample 是完整激光掃描週期要生成的激光光線數量，resolution 乘以 sample 是範圍內數據點的數量

接著要來進行自動避障。在.sdf 檔的相同路徑下用 C++建立以下的 lidar_node.cc 檔：

```cpp
#include <ignition/msgs/twist.pb.h>
#include <ignition/msgs/laserscan.pb.h>
#include <ignition/transport/Node.hh>


std::string topic_pub = "/cmd_vel";   //publish to this topic
ignition::transport::Node node;
auto pub = node.Advertise<ignition::msgs::Twist>(topic_pub);

////////////////////////////////////////////////
/// \brief Function called each time a topic update is received.
void cb(const ignition::msgs::LaserScan &_msg)
{
  ignition::msgs::Twist data;

  bool allMore = true;
  for (int i = 0; i < _msg.ranges_size(); i++)
  {
    if (_msg.ranges(i) < 1.0)
    {
      allMore = false;
      break;
    }
  }
  if (allMore) //if all bigger than one
  {
    data.mutable_linear()->set_x(0.5);
    data.mutable_angular()->set_z(0.0);
  }
  else
  {
    data.mutable_linear()->set_x(0.0);
    data.mutable_angular()->set_z(0.5);
  }
  pub.Publish(data);
}

////////////////////////////////////////////////
int main(int argc, char **argv)
{
  std::string topic_sub = "/lidar";   // subscribe to this topic
  // Subscribe to a topic by registering a callback.
  if (!node.Subscribe(topic_sub, cb))
  {
    std::cerr << "Error subscribing to topic [" << topic_sub << "]" << std::endl;
    return -1;
  }

  // Zzzzzz.
  ignition::transport::waitForShutdown();

  return 0;
}
```

此程式碼定會使模擬開始的時候讓車子往前跑，快碰到牆壁的時候左轉去避障

接著從此連結下載 CMakeLists.txt 檔，並同樣放在與.sdf 檔相同路徑，接著

在.sdf 檔的路徑開啟終端機並下達以下指令：

```
$ mkdir build
```

$ cd build

$ cmake ..

$ make lidar_node

這樣自動避障就建立好了，透過終端機開啟模擬，再開啟另一終端機下達指令：$ ./build/lidar_node，按下播放鑑後車子會自動向前並進行左轉避障。

接著我們把兩種指令透過 Gazebo launch 結合，這樣下達一個指令就能開始避障了。新增如下的 sensor_launch.gzlaunch 檔在.sdf 檔的路徑下：

```
<?xml version='1.0'?>
<gz version='1.0'>
  <executable name='sensor-world'>
    <command>gz sim sensor_tutorial.sdf</command>
  </executable>

  <executable name='lidar_node'>
    <command>./build/lidar_node</command>
  </executable>

</gz>
```

上方檔案需更改成你的.sdf 檔名，接著下達指令就能開啟：

$ gz launch sensor_launch.gzlaunch

註：要關閉軟體請從終端機 Ctrl + C

§ 附錄

1. Gazebo garden 教學 git：
   https://github.com/gazebosim/docs/tree/master/garden/tutorials
2. Ign Gazebo 教學影片參考(舊版)：
   https://www.youtube.com/watch?v=48TX-XJ14Gs&list=PL6FI-gIL5jiEd4Hv-NIAuO2Cbbs27UpAM&pp=iAQB
3. Gazebo garden 官網教學：
   https://gazebosim.org/docs/garden/tutorials
4. Publisher tutorial：
   https://gazebosim.org/api/transport/9.0/messages.html
5. Mass Moment Of Inertia Calculator：
   https://amesweb.info/inertia/mass-moment-of-inertia-calculator.aspx